DotNET 4all by Xander Zelders, http://www.dotnet4all.com

SQL Server developers factsheet

Data types

type	Size	Range (from/to)
Exact numerics		
bigint	8 bytes	-9,223,372,036,854,775,808 9,223,372,036,854,775,807
bit	1 bit	0 to 1
decimal		-10^38 +1 to 10^38 –1
int	4 bytes	-2,147,483,648 to 2,147,483,647
money	8 bytes	-922,337,203,685,477.5808 +922,337,203,685,477.5807
numeric	19 bytes	-10^38 +1 to 10^38 –1
smallint	2 bytes	-32,768 to 32,767
smallmoney	4 bytes	-214,748.3648 to +214,748.3647
tinyint	1 byte	0 to 255
Approximate numerics		
float	8 bytes	-1.79E + 308 to 1.79E + 308
real	4 bytes	-3.40E + 38 to 3.40E + 38
Dates		
datetime	8 bytes	Jan 1, 1753 to Dec 31, 9999
smalldatetime	4 bytes	Jan 1, 1900 to Jun 6, 2079

Type / performance	Characteristics
Character Strings	
char	Fixed-length non-Unicode character. Max 8000 characters
varchar	Variable-length non-Unicode data. Max 8000 characters
varchar(max)	Variable-length non-Unicode data Max 2^31 characters (SQL 2005)
text	Variable-length non-Unicode data. Max 2,147,483,647 characters
Unicode Character Strings	
nchar	Fixed-length Unicode data. Max 4000 characters
nvarchar	Variable-length Unicode data. Max 4000 characters
nvarchar(max)	Variable-length Unicode data Max 2^30 characters (SQL 2005)
ntext	Variable-length Unicode data. Max 1,073,741,823 characters
Binary Strings	
binary	Fixed-length binary data. Max 8000 bytes
varbinary	Variable-length binary data. Max 8000 bytes
varbinary(max)	Variable-length binary data. Max 2^31 bytes (SQL 2005)
image	Variable-length binary data. Max 2,147,483,647 bytes.
Other types	
cursor	A data type for variables or stored procedure OUTPUT parameters that contain a reference to a cursor.
sql_variant	A data type that stores values of various SQL Server 2005-supported data types, except text, ntext, image, timestamp, and sql_variant.
table	Is a special data type that can be used to store a result set for processing at a later time.
timestamp	Is a data type that exposes automatically generated, unique binary numbers within a database.

String Functions (T-SQL)

- **ASCII** (*character*) : Returns the ASCII code value of the leftmost character of *character*
- **CHAR** (*int*) : Converts the integer ASCII code *int* to a character
- **CHARINDEX** (*search*, *expression*, [*start*]) : Returns starting position (int) of first occurrence of the string *search* within table or string *expression* starting from position *start*
- **DIFFERENCE** (*expression1*, *expression2*) : Returns the difference between the SOUNDEX values of the two character expressions as an integer
- **LEFT** (*expression*, *int*): Returns part of character string *expression* starting at *int* characters from the left.
- **LEN** (*expression*) : Returns the number of characters of the string *expression*, excluding trailing blanks.
- **LOWER** (*expression*) : Returns character expression after converting uppercase string to lowercase
- **LTRIM** (*expression*) : Returns a character string after removing all leading blanks.
- **NCHAR** (*int*) : Returns the Unicode character with the given integer code.
- **PATINDEX** ('%pattern%', *expression*) : Returns starting position of the first occurrence of a pattern in a specified expression, or zeros if the pattern is not found, on all valid text and character data types.
- **REPLACE** (*expression1*,*expression2*,*expression3*) : Replaces all occurrences of the second given string expression in the first string expression with a third expression.
- **QUOTENAME** (*character_string*[, *quote_character*]) : Returns a Unicode string with the delimiters added to make the input string a valid Microsoft® SQL Server™ delimited identifier.
- **REPLICATE** (*expression*, *int*) : Repeats a character expression a specified number of times
- **REVERSE** (*expression*) : Returns the reverse of a character expression.
- **RIGHT** (*expression, int*): Returns part of character string *expression* starting at *int* characters from the right.
- **RTRIM** (*expression*) : Returns a character string after removing all trailing blanks.
- **SOUNDEX** (*expression*) : Returns a four-character (SOUNDEX) code.
- **SPACE** (*int*) : Returns a string of *int* spaces.
- **STR** (*float_expression*[, *length*[, *decimal*]]) : Returns character data converted from numeric data.
- **STUFF** (*expression1, start, length, expression2*) : Deletes a specified length (*length*) of characters from *expression1* and inserts another set (*expression2*) at a specified starting point (*start*) of *expression1*.
- **SUBSTRING** (*expression, start, length*) : Returns part of character, binary, text *expression* or image expression starting from position *start* with length *length*
- **UNICODE** (*char*) : Returns the Unicode int value for the first character of *char*.
- **UPPER** (*expression*) : Returns a character expression after converting lowercase string to uppercase.

System Functions (T-SQL)

- **@@ERROR** : Returns the error number for the last Transact-SQL statement executed.
- **@@IDENTITY** : returns the last-inserted identity value.
- **@@ROWCOUNT** : Returns the number of rows affected by the last statement.
- **@@TRANCOUNT** : Returns the number of active transactions for the current connection.
- **APP_NAME** : Returns the application name for the current session if set by the application.
- **CASE** : Evaluates a list of conditions and returns one of multiple possible result expressions.
- **CAST** (*expression* AS *data_type*) / **CONVERT** : Converts an expression of one data type to another.
- **COALESCE** (*expression* [,...*n*]) : Returns the first nonnull expression among its arguments.
- **CURRENT_TIMESTAMP** : Returns the current date and time. ANSI SQL equivalent to GETDATE.
- **CURRENT_USER** : Returns the name of the current user. Equivalent to USER_NAME().
- **DATALENGTH** (*Expression*) : Returns the number of bytes used to represent any expression.
- **FORMATMESSAGE** (*msg_number* , [*param_value* [,...n]]) : Constructs a message from an existing message in sys.messages and returns the formatted message for further processing.
- **GETANSINULL** : Returns the default nullability for the database for this session.
- **HOST_ID** : Returns the workstation identification number.
- **HOST_NAME** : Returns the workstation name.
- **IDENT_INCR** : Returns the increment value (returned as numeric (@@MAXPRECISION,0)) specified during the creation of an identity column in a table or view that has an identity column.
- **IDENT_SEED** : Returns the seed value (returned as numeric(@@MAXPRECISION,0)) that was specified when an identity column in a table or a view that has an identity column was created.
- **IDENTITY** : to insert an identity column into a new table
- **ISDATE** (*expression*): Determines whether an input expression is a valid date.
- **ISNULL** (*expression* , *replacement_value*) : Replaces NULL with the specified value.
- **ISNUMERIC** (*expression)*: Determines whether an expression is a valid numeric type.
- **NEWID** : Creates a unique value of type uniqueidentifier.
- **NULLIF** (*expression* , *expression*) : Returns a null value if the two specified expressions are equal.
- **PARSENAME** ('*object_name*',*object_piece*) : Returns the specified part of an object name. Parts of an object that can be retrieved are the object name, owner name, database name, and server name.
- **PERMISSIONS** ([*objectid* [,'*column*']]): Returns a value containing a bitmap that indicates the statement, object, or column permissions of the current user.
- **SESSION_USER** : returns the user name of the current context in the current database.
- **STATS_DATE** : Returns the date that the statistics for the specified index were last updated.
- **SYSTEM_USER** : Allows a system-supplied value for the current login to be inserted into a table when no default value is specified.
- **USER_NAME** ([*ID*]): Returns a database user name from a specified identification number.

Date and Time functions (T-SQL)

- **DATEADD** (*datepart* , *number*, *date*): Returns a new datetime value based on adding an interval to the specified date.
- **DATEDIFF** (*datepart* , *number*, *date*): Returns the number of date and time boundaries crossed between two specified dates.
- **DATENAME** (*datepart* , *date*): Returns a character string representing the specified datepart of the specified date.
- **DATEPART** (*datepart* , *date*): Returns an integer that represents the specified datepart of the specified date.
- **DAY** (*date*): Returns an integer representing the day datepart of the specified date.
- **GETDATE** : Returns the current system date and time.
- **MONTH** (*date*): Returns an integer that represents the month part of a specified date.
- **YEAR** (*date*): Returns an integer that represents the year part of a specified date.

Dateparts

Datepart	Abbreviations
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dd
hour	hh
minute	mi, n
second	ss, s
millisecond	ms

Cursor Functions (T-SQL)

- **@@CURSOR_ROWS**
  Returns the number of qualifying rows currently in the last cursor opened on the connection.
- **@@FETCH_STATUS**
  Returns the status of the last cursor FETCH statement issued against any cursor currently opened by the connection.
- **CURSOR_STATUS**
  A scalar function that allows the caller of a stored procedure to determine whether or not the procedure has returned a cursor and result set for a given parameter.

Mathematical Functions (T-SQL)

- **ABS** (*Expression*): Returns the absolute (positive) value of a numeric expression.
- **ACOS** (*Expression*): Returns the angle, in radians, whose cosine is the specified float expression; also called arccosine.
- **ASIN** (*Expression*): Returns the angle, in radians, whose sine is the specified float expression. This is also called arcsine.
- **ATAN** (*Expression*): Returns the angle in radians whose tangent is a specified float expression. This is also called arctangent.
- **ATN2** (*Expression*): Returns the angle, in radians, between the positive x-axis and the ray from the origin to the point (y, x), where x and y are the values of the specified float expressions.
- **CEILING**(*Expression*): Returns the smallest integer greater than, or equal to, the specified numeric expression.
- **COS** (*Expression*): Returns the trigonometric cosine of the specified angle, in radians.
- **COT** (*Expression*): Returns the trigonometric cotangent of the specified angle, in radians.
- **DEGREES** (*Expression*): Returns the corresponding angle in degrees for an angle specified in radians.
- **EXP** (*Expression*): Returns the exponential value of the specified float expression.
- **FLOOR** (*Expression*): Returns the largest integer less than or equal to the specified numeric expression.
- **LOG** (*Expression*): Returns the natural logarithm of the specified float expression.
- **LOG10** (*Expression*): Returns the base-10 logarithm of the specified float expression.
- **PI** : Returns the constant value of PI.
- **POWER** (E*xpression, y*): Returns the value of the specified expression to the specified power.
- **RADIANS** (*Expression*): Returns radians of the numeric expression, in degrees.
- **RAND** : Returns a random float value from 0 through 1.
- **ROUND** (*numeric_expression* ,*length* [,*function*]): Returns a numeric value, rounded to the specified length or precision.
- **SIGN** (*Expression*): Returns the positive (+1), zero (0), or negative (-1) sign of the specified expression.
- **SIN** (*Expression*): Returns the trigonometric sine of the specified angle, in radians, and in an approximate numeric, float, expression.
- **SQRT** (*Expression*): Returns the square root of the specified float value.
- **SQUARE** (*Expression*): Returns the square of the specified float value.
- **TAN** (*Expression*): Returns the tangent of the input expression.

SELECT (T-SQL)

The full syntax of the SELECT statement is complex, but the main clauses can be summarized as:

```
SELECT [DISTINCT] [{TOP int | TOP int PERCENT}]
columns
[INTO new_table]
FROM table_source
[[INNER |{{ LEFT | RIGHT | FULL }[ OUTER ]}] JOIN
table_source2 ON table_source.primairy_key =
table_source2.foreign_key][,...n]
[WHERE search_condition]
[GROUP BY group_by_expression]
[HAVING search_condition]
[ORDER BY order_expression [ASC | DESC]]
```

- **DISTINCT** : Specifies that only unique rows can appear in the result set. Null values are considered equal for the purposes of the DISTINCT keyword
- **TOP n [PERCENT]** : Specifies that the first n rows are to be output from the query result set. If PERCENT is also specified, the first n percent are output.
- **INTO new_table**: Creates a new table and inserts the resulting rows from the query into it
- **GROUP BY** : Specifies the groups into which output rows are to be placed and, if aggregate functions are included in the SELECT clause <select list>, calculates a summary value for each group.
- **HAVING** : Specifies a search condition for a group or an aggregate

UPDATE (T-SQL)

The full syntax of the UPDATE statement is complex, but the main clauses can be summarized as:

```
UPDATE table_name
SET column_name = {expression | DEFAULT | NULL}
[,...n]
[WHERE <search_condition>]
```

DELETE (T-SQL)

The full syntax of the DELETE statement is complex, but the main clauses can be summarized as:

```
DELETE [FROM] table_name
[WHERE <search_condition>]
```

INSERT (T-SQL)

INSERT adds a new row to an existing table or a view.
The full syntax of the INSERT statement is complex, but the main clauses can be summarized as:

```
INSERT [INTO] table_name
[(column_list)]
VALUES ({ DEFAULT | NULL | expression }[,...n])
```

CREATE TABLE (T-SQL)

Creates a new table. The full syntax is complex, but the main clauses can be summarized as:

```
CREATE TABLE
[{database_name.[owner].| owner.}] table_name
({<column_definition> | column_name AS
computed_column_expression | <table_constraint>}
[,...n])
```

ALTER TABLE (T-SQL)

Modifies a table definition by altering, adding or dropping columns and constraints, or by disabling or enabling constraints and triggers. The full syntax is complex, but the main clauses can be summarized as:

```
ALTER TABLE table
{[ALTER COLUMN column_name {new_data_type
          [(precision[, scale] )][NULL | NOT NULL]
            | {ADD | DROP} ROWGUIDCOL}] | ADD
          {[ <column_definition> ]|  column_name AS
          computed_column_expression}[,...n]
            | [WITH CHECK | WITH NOCHECK] ADD
          { <table_constraint> }[,...n] | DROP {[CONSTRAINT]
          constraint_name
            | COLUMN column}[,...n] | {CHECK | NOCHECK}
CONSTRAINT
          {ALL | constraint_name[,...n]} | {ENABLE | DISABLE}
TRIGGER   {ALL | trigger_name[,...n]}}
```

Creating / Altering other objects

Stored procedures:

```
CREATE PROCEDURE <name> AS <sql_statement>
ALTER PROCEDURE <name> AS <sql_statement>
DROP PROCEDURE <name>
```

Indexes:

```
CREATE INDEX <name> ON <table> (<column>)
CREATE UNIQUE CLUSTERED INDEX <name> ON <table>.<column>
DROP INDEX <table>.<name>
```

Views:

```
CREATE VIEW <name> [(<Column1>,…)] AS <SELECT_statement>
ALTER VIEW <name> [(<Column1>,…)] AS <SELECT_statement>
DROP VIEW <name>
```

Triggers:

```
CREATE TRIGGER <name> ON <table> FOR INSERT, UPDATE,
          DELETE AS <sql_statement>
ALTER TRIGGER <name> ON <table> FOR UPDATE, DELETE AS
          <sql_statement>
DROP TRIGGER <name>
```

Functions:

```
CREATE FUNCTION <name> RETURNS <data_type> AS
          RETURN <sql_expression>
CREATE FUNCTION <name> RETURNS <data_type> AS BEGIN
          <sql_statement> RETURN <sql_expression> END
ALTER FUNCTION <name> RETURNS <data_type> AS
          RETURN <sql_expression>
DROP FUNCTION <name>
```

Sargability

SQL Server only uses indexes for colums used in sargable expressions. Green = fastest expression.

Sargable	Non-sargable
=	IS NULL
>	<>
<	!=
>=	!>
<=	!<
EXIST	NOT
IS	NOT EXIST
IN	NOT IN
BETWEEN	NOT LIKE
LIKE 'abc%'	LIKE '%abc'
	LIKE '%abc%'
	function on column
	column1 = column1
	column1 = column2

Factsheet by
Xander Zelders
http://www.dotnet4all.com

Checklist for fast queries

- Avoid non-sargable WHERE-clauses. If possible rewrite them to sargable ones
- In the WHERE-clause use the least likely true AND expression first
- Avoid using OR in the WHERE-clause if not all colums have an index
- Avoid using UNION if UNION ALL also does the trick
- Avoid using UNION of two subsets from the same table. Instead use OR in the WHERE-clause
- Avoid using SELECT * FROM when only a few columns are needed. Try to specify each column
- Avoid using COUNT(*) to check the existence of a record. Instead use EXIST
- Always try to use a WHERE-clause in your query to narrow the results
- Try to use the best performing operator as possible
- Avoid using NOT IN. Instead use EXIST, NOT EXIST, IN or LEFT OUTER JOIN with a check for a NULL condition
- Avoid using IN when EXISTS is also possible
- Avoid using IN when BETWEEN is also possible
- In case using IN try to order the list of values so that the most frequently found values are placed first
- Avoid using SUBSTRING in the WHERE-clause. If possible use LIKE instead
- Sometimes consider rewriting a query using a OR to multiple queries combined with a UNION ALL
- Don't use ORDER BY if you don't really need it
- Keep the width and/or number of sorted columns to the minimum
- Keep the number of rows to be sorted to a minimum
- When sorting a specific column often conside making that column a clustered index
- In case of using HAVING try to minimize the amount of rows using a WHERE clause
- In case using LIKE on CHAR of VARCHAR colums quite often consider using the full-text search option
- In case using GROUP BY without an aggregate function try using DISTINCT instead
- Avoid using variables in a WHERE clause in case the query is located in a batch-file

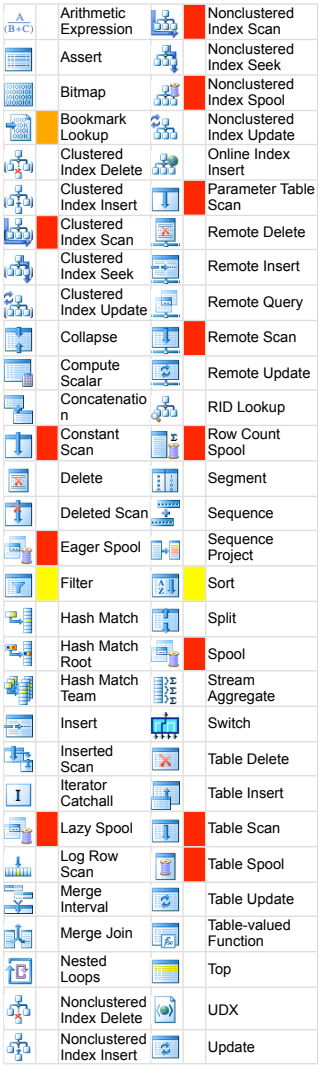
Checklist for creating indexes

- Create indexes on the highly selective colums that are used in the WHERE-clause
- Create indexes on all columns that are used in the WHERE clause in case OR is used
- Create at least a clustered index on every table. Generally use the column that monotonically increases
- Create indexes columns that are frequently accessed by WHERE, ORDER BY, GROUP BY, TOP and DISTINCT
- Only add indexes that will be used frquently
- Avoid adding too much indexes on dynamic tables (subject to many INSERTs, UPDATEs or DELETEs)
- For static tables use a FILLFACTOR and PAD_INDEX of 100. For dynamic tables use a lower FILLFACTOR
- To identify additional indexes use the SS Profiler Create Trace Wizard and trace "Identify Scans of Large Tables"
- Avoid adding indexes twice

Connectionstring

- SQL Server, Standard security: `Provider=sqloledb;Data Source=myServerAddress; Initial Catalog=myDataBase;User Id=myUsername;Password=myPassword;`
- SQL Server, Trusted connection: `Provider=sqloledb;Data Source=myServerAddress; Initial Catalog=myDataBase;Integrated Security=SSPI;`
- SQL Server 2005, Standard security: `Driver={SQL Native Client}; Server=myServerAddress;Database=myDataBase;Uid=myUsername;Pwd=myPassword;`
- SQL Server 2005, Trusted connection: `Driver={SQL Native Client}; Server=myServerAddress;Database=myDataBase;Trusted_Connection=yes;`

Execution Plan Icons (2005)



Red: **Temporary tables/spools**
Try rewriting the query
Red: **Index or table scans**
Create additional or better indexes
Orange: **Bookmark lookups**

- Consider changing the clustered index
- Consider using a covering index
- Consider limiting the number of columns in the SELECT statement.

Yellow: Filter and/or sort

- Consider removing any functions in the WHERE clause
- Consider not using views in your code
- Consider using additional indexes
- Consider not to sort